Designing a Secret Handshake: Authenticated Key Exchange as a Capability System

Dominic Tarr

June 12, 2015

Abstract

(abstract gets written last)

1 Introduction

The development of key exchange algorithms marked the dawn of modern cryptography[5]. Their development was motivated by the desire for secure communications between two parties—yet designing a practical and secure protocol for exchanging a shared key between two authenticated parties is non-trivial[6].

Much of the research into key exchange has produced whole "families" of protocols [8]. Protocols currently in widespread use tend to be layered and configurable (TLS, SSH). This is not to the benefit of application developers—gaining a sufficiently nuanced understanding of such cryptosystems takes considerable study. Providing the developer with *more options* is to provide them with *more opportunities* to add critical security flaws to their application. Thus, recent thought has steered towards providing simple constructions with security properties that are easy to understand and use safely [2]. We apply this philosophy to the design of an authenticated key exchange - a secret handshake. Since a key exchange can be designed with so many possible properties, we adopt the framework of capability systems[11] and allow that to drive the design.

Cryptographic primitives such as a public or secret key are regarded as read and write capabilities, respectively. TahoeLAFS[12] is the inspiration

for this decision. We find that no currently available handshake protocol adequately meets the needs of a capability system. Interestingly, a capability system demands a higher degree of privacy than is provided by other available protocols.

We will describe various key exchange protocols and examin why they are not suitable for a capabality system. In the body of this paper we will describe the protocols discussed abstractly, with sufficient details to persuade the reader of their cryptographic protocols. In some cases this will ignore details from their actual implementations. Finally, we describe our capability based key exchange protocol. In the appendix we have provided a description of our secret-handshake with sufficient detail to produce a compatabile implementation.

2 Notation

$A \rightarrow B$ signifies a message from the client to the server, and $A \leftarrow B$ signifies a response. If A or B is replaced with ?, that indicates that party is not yet authenticated, to the knowledge of the receiver. So for example, Bob sends a message to Alice proving his identity but does not know who alice is yet, that is represented as $? \leftarrow B$.

Long term keys are represented as capitals, and ephemeral keys are represented in lower case. $a \cdot b$ denotes a shared key derived from a and b. When this is used as the key to an encryption, it will be used in the subscript. $box_{[a \cdot b]}$. Note that *box* is authenticated encryption, so this construction has the properties of a mac as well as of encryption. | denotes concatenation.

Finally, it is implied that whenever an actor receives a signed, encrypted or authenticated message they attempt to verify and decrypt it immediately and if this fails then the handshake is immediately aborted.

3 Prior Art

3.1 Authenticated Key Exchange - Station to Station

Here I will describe the STS protocol[6], but this design also forms the basis of most popular protocols (TLS, SSH) except without encrypting the keys or signatures (remember that *box* is authenticated so $box_k(msg)$ can be replaced with $mac_k(msg)$, and it will still constitute a proof of possession of the secret.

$$
\begin{aligned}
? &\rightarrow ? \ : a_p \\
? &\leftarrow B : b_p, Box_{[a \cdot b]}(B_p | Sign_B(a_p | b_p)) \\
A &\rightarrow B : Box_{[a \cdot b]}(A_p | Sign_A(a_p | b_p)) \\
A &\leftarrow B : Box_{[a \cdot b]}(okay)
\end{aligned}
$$

Alice sends a fresh ephemeral key to Bob, who creates one too, signs both keys and sends them back with his public key. Neither party can be assured of the freshness of a message unless it is a cryptographic response to a value they know is fresh, i.e. that they just created. Hence is it no advantage for Alice to send her identity in the first pass, Bob cannot be sure it truly her until the third pass at the earliest.

However, Alice can know Bob's first message is fresh, so STS and many other protocols send the server authentication in the second message. To resist an identity misbinding attack we require proof that the other party possesses the shared secret $a \cdot b$. Constructing $box_{[a \cdot b]}$ or a mac accomplishes that [8, section 3.1]

Often in the description of a handshake protocol ends as soon as it's possible for Alice and Bob to send each other a encrypted message. However, if messages $A \rightarrow B$ and $A \leftarrow B$ have not both been received then at least one party does not know whether they are authenticated. When this is the case, they do not know for sure that they are authenticated until they receive the first encrypted message in the main body of the communication session. For this reason I've represented STS as a 4 pass protocol, although the original paper describes it as 3 pass.

Is STS suitable for a capability system? No. The first thing STS does is authenticate the server, but the first question a capability system should ask is whether the client has the capability sufficient to access this resource.

By authenticating Bob first, to an unauthenticated client, the public keys are leaked to anyone who connects to Bob and executes the protocol correctly.

3.2 Encrypted Authenticated Key Exchange

CurveCP[1] performs authentication based entirely on nacl's[2] *box* primitive, which uses curve25519 keys. curve25519 keys cannot be used to create signatures, but are combined via scalar multiplication to produce shared keys.

recall $Box_{[a \cdot b]}(content)$ creates an authenticated, encrypted message that can be opened by someone who knows either (a_{secret}, b_{public}), or (a_{public}, b_{secret}). I have represented this with a $\cdot$ to remind the reader that this operation is undirected. A box is between two keys, and not from one key to another – unlike RSA encryption.

$$
\begin{aligned}
? \rightarrow ? &: a_p, Box_{[a \cdot B]}(okay) \\
? \leftarrow B &: Box_{[a \cdot B]}(b_p) \\
A \rightarrow B &: Box_{[a \cdot b]}(A_p || Box_{[A \cdot B]}(a_p)) \\
A \leftarrow B &: Box_{[a \cdot b]}(okay)
\end{aligned}
$$

CurveCP does have one feature that suggests it's suitability for use in a capability system - for Alice to authenticate to Bob she must already know his public key. Knowledge of the Bob's public key thus forms a capability to connect to Bob.

Unfortunately, this construction has two problems.

1. because the first message is encrypted, a replay attacker may use it to confirm the identity of a server after it has moved to a different address, if it still uses the same key as before. Since only the owner of that key can decrypt that packet, if they respond it confirms they have the same identity. This reveals information that a replay attacker doesn't deserve. Having this information could motivate the replay attacker to look for other weaknesses in Bob, thus in the interest of easy to understand security properties this small leak must be closed. The simplest change to mitigate this attack to CurveCP would be to require the server to respond with nonsense to an incorrect initial packet, but this information would probably still be available via timing information and thus is contrary to the design philosophy of nacl[2]

2. The simple way that CurveCP uses key sharing for authentication enables Key Compromise Impersonation by an attacker who gains possession of B_{secret} [4]. If Conrad gains B_{secret} he can and impersonate Alice to Bob. Condrad would connect to Bob, create an ephemeral key a, box it to Bob, and when Bob responds, create $Box_{[a \cdot b]}(A_p || Box_{[A \cdot B]}(a_p))$, except using (A_{public}, B_{secret}), as Bob would when opening the box, not

(A_{secret}, B_{public}) as Alice would have when sealing it. If Conrad possesses B_{secret} he can impersonate arbitary keys to Bob. Authenticating as Alice in a capability system must only be possible via the possession of A_{secret} - the capability to Alice's identity.

3.3 Deniable Authenticated Key Exchange

There is a class of key exchange protocols which make deniability a design goal[3, 9, 10]. The argument for this is that when you engage in casual communication you do not create evidence that you said what you did.

The currently available protocols are unsuitable for a capability system for the same reason as STS is unsuitable– the transmission of Bob's long term key to an unauthenticated client. This would be a simple fix, and whether a deniable key exchange is suitable for a capability system deserves closer study. Since TextSecure[9] supercedes OTR[3], but uses a 3rd party introducer and is thus not directly analogous to a typical secure channel we will examine the noise[10] protocol first.

Here we will extend the notation for shared keys to express keys shared between multilpe pairs of keys. $a \cdot b | a \cdot B$ denotes the hash of $a \cdot b$ concatenated with $a \cdot B$. Only a party that can construct both the component keys can construct the composite key.

$$
\begin{aligned}
? &\rightarrow ? &&: a_p \\
? &\leftarrow B &&: b_p, Box_{[a \cdot b]}(B_p), Box_{[a \cdot b | a \cdot B]}(okay) \\
A &\rightarrow B &&: Box_{[a \cdot b]}(A_p), Box_{[a \cdot b | A \cdot b]}(okay) \\
A &\leftarrow B &&: Box_{[a \cdot b | a \cdot B | A \cdot b]}(a_p | b_p)
\end{aligned}
$$

This is a better design, but it has some features that make it less suitable for a capability system. Note that although it is specified as a 3 pass protocol the client does not know it is authorized until it receives the first encrypted message from the server, so again, it is really a 4 pass protocol. Note that unlike CurveCP, a shared key is not derived between long term keys, but instead only between an ephemeral key and a long term key. Even if Alice suspects that Conrad may have compromised her long term key, A, she trusts that he surely cannot know her ephemeral key, a. Without knowing a_{secret} Conrad cannot construct $a \cdot B$, unless it really *is* Bob. To provide the same

assurance to Bob, they end up with three way key $a \cdot b | a \cdot B | A \cdot b$, as in TextSecure[9].

1. The handshake is protected from eavesdroppers – but anyone who connects to the server will be sent the public key, so we cannot use knowledge of the server's key as a capability.

2. KCI is still possible, but harder – To impersonate an arbitary key to Bob you have to know Bob's ephemeral *and* long term secret keys. This would be possible for anyone who had passive read access to Bob's memory – at first glance this may seem like a unlikely proposition, but in fact if Bob is running on a rented virtual machine that is precisely the situation he would be in.

Problem 2 would be avoided if the protocol was only ever run on physical hardware–but this is completely unreasonable. Since it's possible to authenticate to Bob as alice by knowing b and B, and not strictly by knowing A, then this protocol fails to form a well behaved capability system. Also, the property of deniability seems difficult to reason about. Will the higher level protocol introduce evidence of the communication? Will the contents stored for a long period of time? In any case, a secure channel is used for a lot more than casual social communications, and deniability does not appear to offer any special advantage to a capability system.

4 A New Design

If we take the general pattern from the noise handshake, but turn it around so that Alice authenticates first, then we start to get something that looks like a capability system.

If Alice *preauthenticates* Bob, then Bob can authenticate Alice using one more pass. With two initial passes to prevent replay attacks, we have a 4 pass protocol. This is no worse than the above, even though we do not authenticate anyone until the third pass.

To "preauthenticate" Bob, Alice sends a proof of both her identity, and her intention to connect to Bob. Preauthentication can be implemented with both encryption and signatures.

$$
\begin{aligned}
? &\rightarrow ? \ : a_p \\
? &\leftarrow ? \ : b_p \\
A &\rightarrow B : Box_{[a \cdot b | a \cdot B]}(A_p) \\
A &\leftarrow B : Box_{[a \cdot b | a \cdot B | A \cdot b]}(okay)
\end{aligned}
$$

Requiring Alice to authenticate first is unusual, but I think this is a fair deal. Bob has already put themself at a disadvantage by allowing himself to be publically addressable. It's only fair that Alice authenticates first. By encrypting her authentication she need not reveal her identity to anyone but the one true Bob. Likewise, if Bob chooses not to accept the call, then Alice won't be able to deduce whether or not it was really Bob. Maybe it was but he did not wish to speak to her, maybe it was just a wrong number. This protects Bob from harassment.

In cases where Bob is not concerned with the identity of his clients, he may simply allow anyone to authenticate, and Alice can generate a second ephemeral identity instead of using A.

In this design Bob's public key forms as an access capability, however it still suffers the KCI problem that noise and TextSecure have.

Authenticated handshakes based on signatures do not have these KCI problems. Key exchange is required for confidentiality and forward security, but signatures are required for simple resistance to KCI. By incorporating a signatures into the handshake we can achive a truly well behaved capability system.

Since we will need both exchange and signing keys, an identity could be represented by a pair of signing and exchange keys. nacl uses ed25519 keys for signatures, and curve25519 keys for exchange. However, nacl also provides a functions to convert signing to exchange keys, so an identity could be represented as signing key. This signing key would be converted to an exchange key when necessary.

$$
\begin{aligned}
? \rightarrow ? &: a_p \\
? \leftarrow ? &: b_p \\
H &= A_p | Sig_A(B_p | hash(a \cdot b)) \\
A \rightarrow B &: Box_{[a \cdot b | a \cdot B]}(H) \\
A \leftarrow B &: Box_{[a \cdot b | a \cdot B | A \cdot b]}(Sig_B(H))
\end{aligned}
$$

The design is getting much better. We resist eavesdropping, replay, man-in-the-middle, and KCI attacks. There are just a few minor niggles to tidy up.

If either of the two initial packets are tampered with, it would be undetected until the first authenticated packet is received, this is not necessarily insecure, but it is poor design. More importantly, if the signing keys are also used elsewhere, it's possible that a signature from this protocol or gets reused elsewhere, or vice versa [7]. These issues can be addressed by adding an application key (K) as a capability to speak this protocol. The ephemeral keys can be authenticated by using the K as the key to an *hmac*. By including K in each signature it is demonstrated that the signature belongs within this protocol. It is vital that if there are any other cases where a signing key is used, then a similar level of care is taken to prevent ambigious interpretations of signatures created.

K_{app} could be the hash of the protocol version and the terms of service, or it simply be a random number.

$$
\begin{aligned}
? \rightarrow ? &: a_p, hmac_K(a_p) \\
? \leftarrow ? &: b_p, hmac_{[K | a \cdot b]}(b_p) \\
H &= A_p | sign_A(K | B_p | hash(a \cdot b)) \\
A \rightarrow B &: box_{[K | a \cdot b | a \cdot B]}(H) \\
A \leftarrow B &: box_{[K | a \cdot b | a \cdot B | A \cdot b]}(sign_B(K | H | hash(a \cdot b)))
\end{aligned}
$$

Alice's authentication, $A_p | sign_A(K | B_p | hash(a \cdot b))$, proves that she possesses A, and that this proof is for this protocol (via K) and for this handshake (via $hash(a \cdot b)$). Incase Bob does not have alice's key yet, she sends

it with her public key.

For Bob to authenticate back to Alice, he could just sign the proof Alice sent him, and send it back. H is already cryptographically linked to the preceding passes, however, it is easier to persuade ourselves that signing K mitigates the Chosen Protocol Attack[7] than that it's unlikely A_p is never the access cap to another protocol.

Alice and Bob can now use their shared secret, $K|a \cdot b|a \cdot B|A \cdot b$, with a bulk encryption protocol to secure a two-way communication channel.

Usually a secret key represents an *identity*, but one of the interesting things in cryptographic capability systems, such as tahoeLAFS is that a secret key more accurately a *write capability*. Systems built upon this secret handshake protocol may have shared secret keys that allow other actors to authenticate to a particular role semianonymously.

5 Future Work

The latency induced by a 4 pass protocol may be prohibitive for some applications. If some mechanism was provided to prearrange a single use key for the next session it may be possible to reduce this to two passes, once a pair of actors have established contact.

Some readers will be wondering how Alice is to learn Bob's public key? There is a large design space, from Certificate Authority style central registries, to lookup via DHTs, to gossip networks or webs of trust. Interestingly, although in this protocol capabilites are made from public keys, they need not be "public". Access to any actor can be restricted via a secret public key. Likewise, if an actor wishes to provide a specific api on an anonymous basis, they may do so by creating a shared private key–this is not an oxymoron in a capability system, it is simply a capability that has been delegated. The use of such capabilitys may be restricted to a limited number of times or within a specific time window.

6 Conclusion

I have described a highly private 4 pass handshake protocol that is suitable for capability systems. It does not suffer from replay, eavesdropping, man in the middle, or key compromise impersonation. It can be described in just a

few lines, and I have a reference implementation which is just a few hundred. Although the objective was a well-behaved capability system it is interesting that this has produced a design with a higher degree of privacy than other key exchange protocols.

References

[1] Daniel Bernstein. Curvecp: Usable security for the internet. http://curvecp.org/index.html, Feb 2011. Accessed: 2015-06-10.

[2] Daniel J. Bernstein, Tanja Lange, and Peter Schwabe. The security impact of a new cryptographic library. In *Proceedings of the 2Nd International Conference on Cryptology and Information Security in Latin America*, LATINCRYPT'12, pages 159–176, Berlin, Heidelberg, 2012. Springer-Verlag.

[3] Nikita Borisov, Ian Goldberg, and Eric Brewer. Off-the-record communication, or, why not to use pgp. In *Proceedings of the 2004 ACM workshop on Privacy in the electronic society*, WPES '04, pages 77–84, New York, NY, USA, 2004. ACM.

[4] CodesInChaos. Curvecp review - part 1 crypto. https://codesinchaos.wordpress.com/2012/09/09/curvecp-1/, Sept 2012.

[5] Whitfield Diffie and Martin E. Hellman. New directions in cryptography, 1976.

[6] Whitfield Diffie, Paul C. van Oorschot, and Michael J. Wiener. Authentication and authenticated key exchanges, 1992.

[7] John Kelsey, Bruce Schneier, and David Wagner. Protocol interactions and the chosen protocol attack. In *In Proc. 1997 Security Protocols Workshop*, pages 91–104. Springer-Verlag, 1997.

[8] Hugo Krawczyk. Sigma: the sign-and-mac approach to authenticated diffie-hellman. In *SIGMA, full version. http://www.ee.technion.ac.il/ hugo/sigma.html*, 2003.

[9] Moxie Marlinspike. Simplifying otr deniability. https://whispersystems.org/blog/simplifying-otr-deniability/, July 2013.

[10] Trevor Perrin. noise. https://github.com/trevp/noise/blob/master/noise.md, Aug 2014.

[11] Tristan Slominski. Towards a universal implementation of unforgeable actor addresses. http://www.dalnefre.com/wp/2013/10/towards-a-universal-implementation-of-unforgeable-actor-addresses/, Oct 2013. Accessed: 2015-06-10.

[12] Zooko Wilcox-O'Hearn and Brian Warner. Tahoe the least-authority filesystem. Cryptology ePrint Archive, Report 2012/524, 2012.